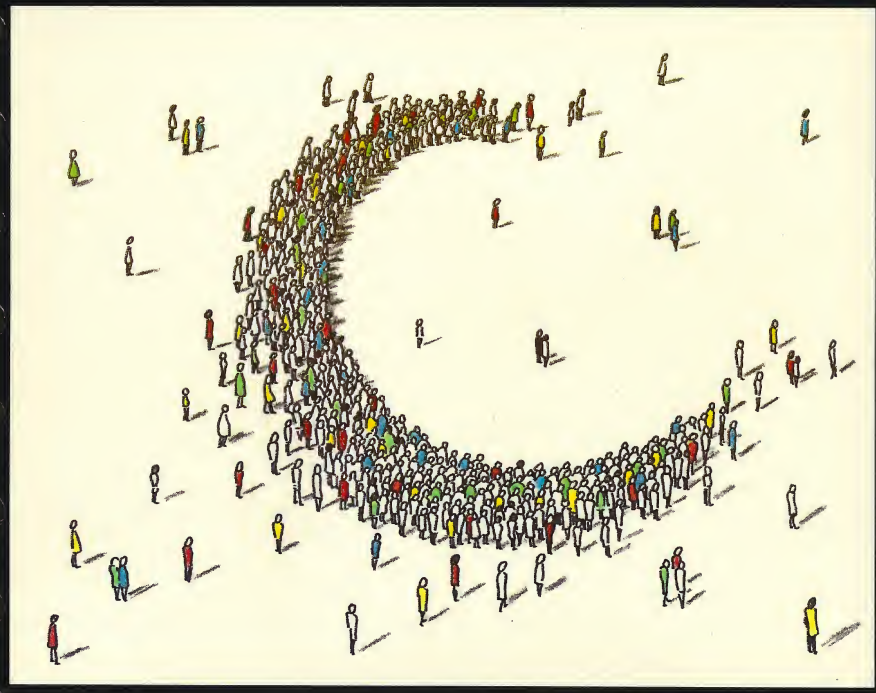
0479,544

ACORNSOFT
The choice of experience in software.

C



C

User Guide

ACORNSOFT
The choice of experience
in software.

This BSC implementation of C was written and designed by Tim Bell, David Christensen, James Mansion and Jim Warwick.

Typeset by Interaction Systems Limited, Cambridge.

British Broadcasting Corporation has been abbreviated to BBC in this publication.

FIRST EDITION

ISBN 1 85250 037 9

Part no 0479,544

Published by Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN.

CONTENTS

INTRODUCTION

The purpose of this manual is to illustrate how to use the Acornsoft C system. It does not teach the C language, but explains the steps necessary to create and compile C programs using the Acornsoft C system.

The manual assumes that you have a knowledge of programming in general and a basic knowledge of the C programming language and its terminology in particular. If you require tutorial information about the C language itself, look at the list of recommended books in **Appendix E**.

THIS MANUAL

The chapters in this manual are organised as follows:

- *The C language* deals with the differences between Acornsoft C and the version defined by Kernighan and Ritchie.
- C *system overview* introduces you to the use of the Acornsoft C system and its editor.
- *The C editor* discusses the C editor in detail and explains how to use it.
- *Compilation* discusses the compilation process.
- *Linking files* and *Executing the object code* tell you how to invoke the linker and execute object code.
- *Header files* describes the header files found in C.
- *Library routines* lists all library routines and gives details on each.
- *Run-time memory allocation* discusses the memory map and run-time system workspace and variables.

- *Appendix A* is an editor command summary.
- *Appendix B* is a summary of all commands.
- *Appendix C* alphabetically lists all library routines, giving a page number for each.
- *Appendix D* contains error messages.
- *Appendix E* is a bibliography containing a list of further reading material on the C programming language.

CONVENTIONS USED

Throughout this manual, the following conventions are used:

- Text that is entered by you, text that appears on the screen, C keywords, library routines, macros and commands are printed in the following style:

  ```
  #Save [current filename]
  ```

- Keytops are shown as you see them on the keyboard, surrounded by a box. For example:

  RETURN

- Combinations of keys are referred to by the key names printed side by side. For example:

  SHIFT f1

  means you must hold down the shift key while simultaneously pressing the f1 key.

THE C LANGUAGE

The Acornsoft C system is based on the version of C described in Kernighan and Ritchie's book *The C Programming Language*. This chapter documents the minor differences between Kernighan and Ritchie's version of C and the Acornsoft C system.

Throughout this chapter, Kernighan and Ritchie's definition is referred to as standard C, although it is not a formal national or international standard.

Because many other implementations of C are also based on Kernighan and Ritchie's definition, you can move C programs freely between different computers, as long as you avoid extensions to the standard peculiar to individual machines. This portability applies between the Acornsoft C system and the C system provided with the Acorn Cambridge Workstation.

Note, however, that standard C does not define a full set of library routines which all systems must provide. The routines that are mentioned are not fully defined.

Acornsoft C provides a comprehensive set of library functions. These routines conform as far as possible to the X/OPEN Standards Group proposals. (See the *X/OPEN Portability Guide* for information on the X/OPEN Standards Group. It is listed in **Appendix E.**) The most significant differences occur in the low-level i/o routines which are constrained by the operating system and filing systems. The higher-level libraries contain routines which are common to almost all implementations of C. However, it is not guaranteed that these routines will have exactly the same effect as their counterparts in other versions of C, particularly in their handling of errors.

RESTRICTIONS

This section discusses Acornsoft C restrictions. They include restrictions on:

- type specifiers
- storage classes
- anachronisms
- the implementation of \n
- the # directives

Type specifiers

The following combinations of type specifiers are allowed. Optional clauses are given in square brackets:

Type	Storage	Range
`char`	1 byte	0 to 255
`int`	2 bytes	−32768 to 32767 (−2 to 2 −1)
`short [int]`	2 bytes	−32768 to 32767 (−2 to 2 −1)
`long [int]`	4 bytes	−2147483648 to 2147483647 (−2 to 2 −1)
`unsigned [int]`	2 bytes	0 to 65535
`[long] float`	5 bytes	−1e38 to 1e38 (approx)
`double`	5 bytes	−1e38 to 1e38 (approx)
`struct`	-	
`union`	-	
`void`	-	

User-defined types are allowed; `typedef` is fully implemented.

Enumerated types are not allowed. The `enum` data type defined in standard C allows you to construct new data types by enumerating the values which variables of that type may take. Similarly, bitfields are not implemented. Bitfields are groups of bits, located next to each other within a single integer.

Storage classes

No use is made of the register storage class. It is defined in standard C as being used to advise the compiler that the variable being declared will be heavily used and, where possible, should be put in a machine register to increase execution speed. In Acornsoft C, it is treated as being equivalent to an auto declaration, since all reasonable optimisation of that nature, whatever the declaration, is performed automatically by the compiler.

Anachronisms

Note the following points regarding anachronistic forms:

- `=<operator>` is illegal. It is used in earlier versions of C instead of `<operator>=` for assignment operators.
- `int X 3` is illegal. It is used in earlier versions of C to initalise a variable. In Acornsoft C it must be written as `int X = 3;`.

\n

In 32016 C and Unix the implementation of \n is ASCII 10. In Acornsoft C \n is ASCII 13.

The # directive

When typing preprocessor directives, do not insert a space between the # symbol and the directive. For example, the directive:

```
# define
```

is illegal. It must be entered as:

```
#define
```

EXTENSIONS

This section discusses Acornsoft C extensions. They include the data type `void`, the removal of restrictions on structure member names and the use of `floats`.

Data type void

Acornsoft C has an extra data type, `void`, which is included in many modern compilers but is not a part of standard C. `void` is a special data type having no value, and is used to indicate that a function returns no value. This allows expressions to be cast to type `void` in order to discard their value explicitly. The (non-existent) value of a `void` expression may not be used in any way and neither explicit nor implicit conversions may be applied to such a value. A `void` expression may be used, therefore, only as an expression statement or as an operand of a comma operator.

Structure member names

Acornsoft C makes no restrictions on the use of the same member name in different structures. This feature occurs in many modern compilers. However, it is not found in standard C. In standard C, the same member name may occur in

different structures only if the fields identified by the member name and all preceding fields are the same.

Floats

Acornsoft C lets you use a `float` in statements which require an integer (eg `switch`). When a `float` is given, the integer part of the `float` is used. For example, `switch (1.0)`, `switch (1.4)` and `switch (1.84320)` are all equal to `switch (1)`.

Note: Use of this feature is not recommended. It should only be used with caution.

C SYSTEM OVERVIEW

The Acornsoft C installation and startup procedures are described in the configuration leaflet accompanying this manual. Make sure C is installed on your system before proceeding. Once C is installed and entered you are ready to type in C commands and operating system commands.

This chapter introduces you to the Acornsoft C system command line interpreter and takes you briefly through the steps required to produce and run an Acornsoft C program.

THE C SYSTEM COMMAND LINE INTERPRETER

C is a compiled language. This means that you cannot type sections of C and have them executed immediately. In this respect, C differs from BASIC, which is 'interactive' and executes immediate statements as soon as they are entered.

In C, only a small set of commands is required. They can be typed in upper or lower case and include the following:

- *CLOSE:* This command closes all open files on the current filing system. It is useful in situations where BREAK is pressed in the middle of a compilation, resulting in the source, object and/or temporary files being left open. The file(s) are inaccessible until closed.

- *COMPILE filename:* This command compiles a C source text file and generates a linkable file. It may take the following options:

Option	Description
`-dmacroname`	define macroname
`-mnumber`	define the maximum number of errors allowed
`-f`	turn function tracing on

Option	Description
`-q`	do not print the compiler title and summary information
`-l`	invoke the linker automatically
`-r<restofline>`	run after linking

These options are discussed in greater detail in the chapter entitled **Compilation**.

- *EDIT filename (sideways RAM) /*EDIT filename (Second Processor):* This command activates the screen editor used to create and amend a C source file. If you specify a filename after `EDIT`, the file specified is loaded before the editor is entered. If a file is already loaded in memory and no filename is specified when the editor starts up, the file currently in memory appears on the screen. If there is no file currently in memory, the editor contains no text.

- *HELP:* This command lists all Acornsoft C commands.

- *LINK filename:* This command links two or more link files together. One of the files is, by default, the standard library. `LINK` generates an object program which may subsequently be executed. The following options are possible:

Option	Description
`-daddress`	generate the code to end at a given memory address (in hexadecimal)
`-l`	do not link against the standard library, `stdlib`
`-n`	give function information

Option	Description
`-q`	do not print the linker title and summary information
`-r<restofline>`	run the object program automatically

- *MODE number:* This command sets the current screen mode.

- *REPORT:* This command causes the last system message encountered to be printed.

- *SETPATH element [, element]:* This command specifies where C looks for the various components of the system, ie C commands, header files, etc. It is generally referred to as 'the current path'.

  The default path values depend on the filing system you are using:

  – DFS `setpath :0`

  – ADFS or NFS `setpath $.C`

  The system returns to default values whenever BREAK is pressed or the C system is re-entered. For example, when it looks for `include` files, it will look in directory `<current path>.h`.

- *SHOWPATH:* This command prints the current path.

- *TRACE:* This is a toggle command relating to the function `trace` facility. When it is on, information about the arguments passed to certain routines and the value returned by them is sent to the standard error file. This information is given for any routines defined in source text files which

were compiled with the `-f` option. When it is off no information is displayed.

- . *(full stop)*: This command specifies that the rest of a line should be treated as a comment and hence ignored. For example:

```
. This is a command line
```

Any other (ie unrecognised) command, such as hello, is interpreted as the name of a code file. C attempts to find it using the current path. If no file is found, C generates the `Not Found` error message.

PRODUCING AND RUNNING A C PROGRAM

This section includes a brief description of how to produce and run a C program. It uses a sample source text file included on the Acornsoft C disc, `C.HelloW`. The description is intended to give you an overall view of the procedures you need to follow. More detailed information is found in the following chapters of this manual.

Producing and running a program is done in four stages:

1 Prepare the source text.

2 Compile the program.

3 Link with the libraries.

4 Execute the program image.

Preparing the source text

The source text for the sample text file `C.HelloW` is as follows:

```
/* HelloWorld */

#include <h.stdio>

main()

{
  puts("Hello World !");
}
```

Compiling the program

To compile a program from this C source text, type:

```
COMPILE HelloW RETURN
```

Do not give the directory name since the compiler automatically searches in directory C for the source text file.

The compiler generates a corresponding code in a form suitable for the linker and saves it to disc (or the storage device used by your filing system) in the file `L.HelloW`. If the compilation is successful, the following message appears on the screen (the numbers on the last line may vary depending on the system you are using):

```
Acornsoft-BSC C Compiler V1.00
Including h.stdio
code: 0x0042 data: 0x000E total: 0x0050
```

If it is not successful, an error message is displayed.